The Persistence Life Cycle

- In this lecture, we discuss the life cycle of persistent objects in Hibernate.

- These persistent objects are POJOs without any special marker interfaces or inheritance related to Hibernate.

- Part of Hibernate's popularity comes from its ability to work with a normal object model.

Introduction to the Life Cycle

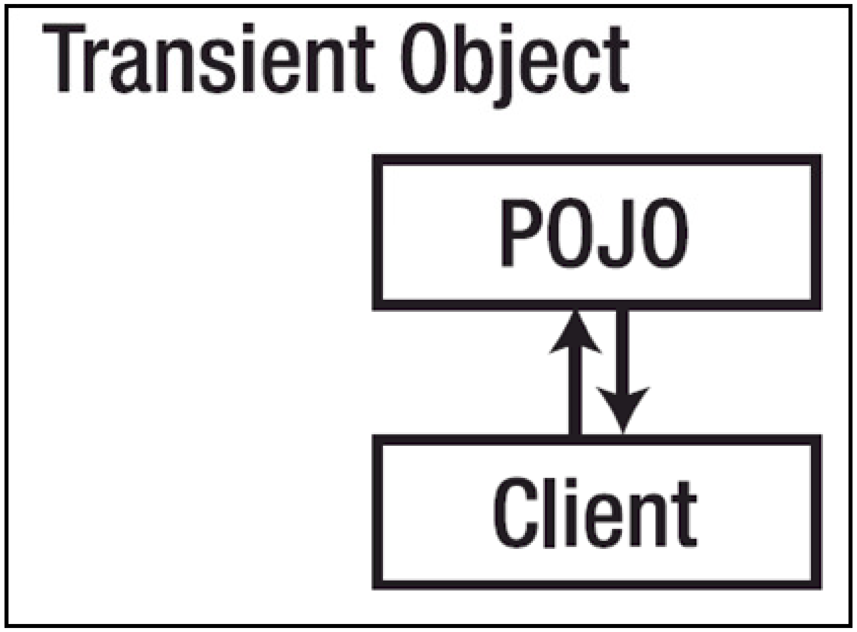
- After adding Hibernate to your application, you do not need to change your existing Java object model to add persistence marker interfaces or any other type of hint for Hibernate.
- Instead, Hibernate works with normal Java objects that your application creates with the new operator, or that other objects create.
- For Hibernate’s purposes, these can be drawn up into two categories:
  - objects for which Hibernate has entity mappings, and
  - objects that are not directly recognized by Hibernate.

- A correctly mapped entity object will consist of fields and properties that are mapped, and that are themselves either references to correctly mapped entities, references to collections of such entities, or “value” types (primitives, primitive wrappers, strings, or arrays of these).

- Given an instance of an object that is mapped to Hibernate, it can be in any one of three different states:
  - transient,
  - persistent, or
  - detached

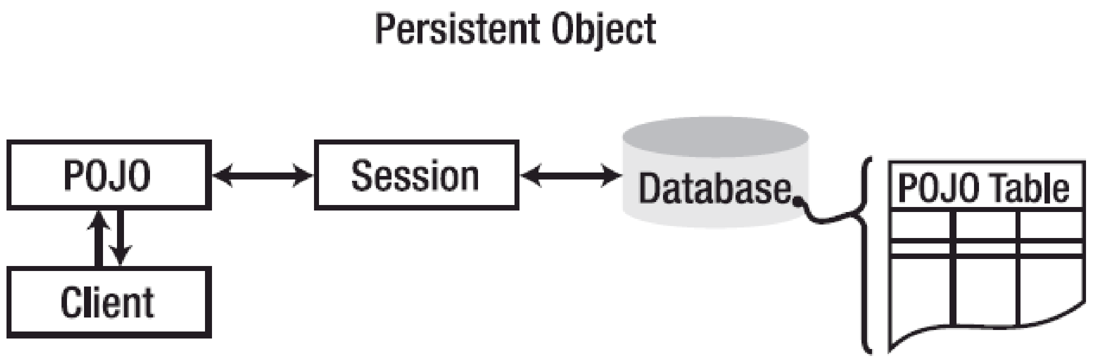
1. Transient objects

- *Transient objects exist in memory, as illustrated in the figure.*
- *Hibernate does not manage* transient objects or persist changes to transient objects.
- To persist the changes to a transient object, you would have to ask the session to save the
- transient object to the database, at which point Hibernate assigns the object an identifier.



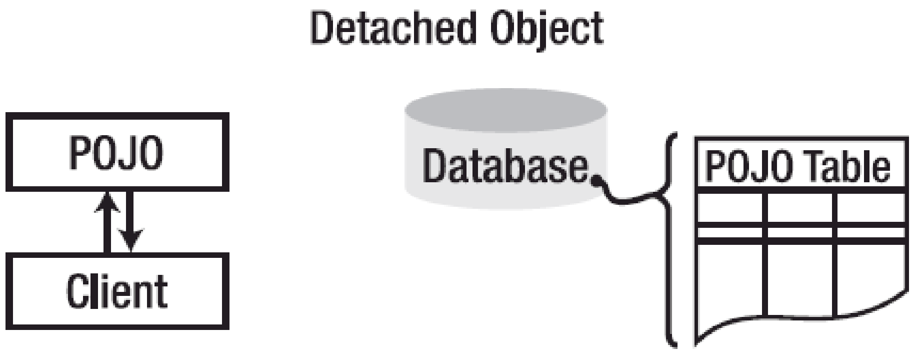
2. Persistent objects

- *Persistent objects exist in the database, and Hibernate manages the persistence for persistent* objects
- This relationship between the objects and the database is shown in the figure.
- If fields or properties change on a persistent object, Hibernate will keep the database representation up-to-date.



3. Detached objects

- *Detached objects have a representation in the database, but changes to the object will not* be reflected in the database, and vice versa.
- This temporary separation of the object and the database is shown in the Figure.
- A detached object can be created by closing the session that it was associated with, or by evicting it from the session with a call to the session's evict() method.



persisting changes made to a detached object

- In order to persist changes made to a detached object, the application must reattach it to a valid Hibernate session.
- A detached instance can be associated with a new Hibernate session when your application calls one of the load(), refresh(), merge(), update(), or save() methods on the new session with a reference to the detached object.
- After the call, the detached object would be a persistent object managed by the new Hibernate session.
- Versions prior to Hibernate 3 had support for the Lifecycle and Validatable interfaces.
- These allowed your objects to listen for save, update, delete, load, and validate events using methods on the object.
- In Hibernate 3, this functionality moved into events and interceptors, and the old interfaces were removed.